Amendments to the Claims

Please amend claims 1, 15, 20, 34, 35, 40, 41 & 51 and add new claims 55-58 as set forth below. All claims are reproduced below, with changes in the amended claims shown by underlining (for added matter) and strikethrough/double brackets (for deleted matter).

1. (Currently Amended) A method of testing a software component comprising multiple layers of software, the method comprising:

creating an abstraction matrix in mathematical abstract form by automatically partitioning the software component into multiple layers, the abstraction matrix comprising state and event information taking into account relationships that exist between the multiple software layers;

parsing the abstraction matrix to automatically generate and factor out doable test cases and mapped expected results therefor;

separating the test cases based on the software layers of the software component, and associating data structures with the separated test cases of the software layers, the data structures allowing the test cases of the various software layers to be uncorrelated;

employing the software component in executable form to generate for each software layer of the software component test case execution threads from the test cases and mapped expected results for that software layer; and

executing in parallel at least some of the test case execution threads for at least one software layer of the software component, thereby testing the software component.

2. (Original) The method of claim 1, wherein the creating of the abstraction matrix comprises creating the abstraction matrix from a functional specification of the software component.

3. (Original) The method of claim 1, wherein the parsing, the separating, the employing and the executing comprise automated processes.

4. (Original) The method of claim 1, wherein the executing comprises executing in parallel at least some of the test case execution threads for multiple layers of the software component.

5. (Original) The method of claim 4, wherein the executing comprises executing in parallel at least some test case execution threads of each layer of the software component.

6. (Original) The method of claim 1, wherein the separating the test cases based on layers of the software component results in each layer of the software component having multiple test cases associated therewith.

7. (Original) The method of claim 1, wherein the creating comprises creating the abstraction matrix so that the abstraction matrix describes each layer of the software component.

8. (Original) The method of claim 7, wherein the creating comprises creating a separate abstraction file for each layer of the layers of the software component.

9. (Original) The method of claim 1, further comprising providing attributes for use in generating the test cases and mapped expected results therefore.

10. (Original) The method of claim 1, wherein the creating comprises creating the abstraction matrix to comprise state information and events that are associated with each state of the software component.

11. (Original) The method of claim 10, wherein the creating comprises creating the abstraction matrix to include current state information and next state information, and wherein for at least one current state the event information comprises an event which leads from that current state to at least one next state.

12. (Original) The method of claim 1, wherein the software component comprises an operating system component.

13. (Original) The method of claim 1, wherein the creating comprises creating the abstraction matrix to identify a minimal number of states required to model the software component.

14. (Original) The method of claim 1, wherein the associating data structures for the separated test cases of the layers comprises associating information on the relationships between layers, thus allowing test cases of the various layers to be uncorrelated.

15. (Currently Amended) A computer-implemented method of generating test cases for use in creating a software component, the software component comprising multiple layers of software, said method comprising:

ascertaining a functional specification of the software component;

creating an abstraction matrix in mathematical abstract form that describes the software component by automatically partitioning the software component into multiple software layers using the functional specification, the abstraction matrix comprising state and event information taking into account relationships that exist between the multiple software layers;

parsing the abstraction matrix to automatically generate and factor out doable test cases and mapped expected results therefor; and

separating the test cases based on the software layers of the software component, and associating data structures with the separated test cases of the software layers, the data structures allowing the test cases of the various software layers to be uncorrelated.

16. (Original) The method of claim 15, wherein the parsing and the separating comprise automated processes.

17. (Original) The method of claim 15, wherein the creating comprises creating a separate abstraction file for each layer of the layers of the software component.

18. (Original) The method of claim 15, wherein the creating comprises creating the abstraction matrix to include current state information and next state information, and wherein for at least one current state the event information comprises an event which leads from that current state to at least one next state.

19. (Original) The method of claim 15, wherein the associating data structures for the separated test cases of the layers comprises associating information on the relationships between layers, thus allowing test cases of the various layers to be uncorrelated.

20. (Currently Amended) A system for testing a software component comprising multiple layers of software, the system comprising:

an abstraction matrix in mathematical abstract form that describes the software component, the abstraction matrix partitioning the software component into multiple software layers and comprising state and event information taking into account relationships that exist between the multiple software layers;

means for parsing the abstraction matrix to automatically generate and factor out doable test cases and mapped expected results therefor;

means for separating the test cases based on the software layers of the software component, and for associating data structures with the separated test cases of the software layers, the data structures allowing the test cases of the various software layers to be uncorrelated;

means for employing the software component in executable form to generate for each software layer of the software component test case execution threads from the test cases and mapped expected results for that software layer; and

means for executing in parallel at least some of the test case execution threads for at least one software layer of the software component, thereby testing the software component.

21. (Original) The system of claim 20, wherein the abstraction matrix is created from a functional specification of the software component.

22. (Original) The system of claim 20, wherein the means for parsing, the means for separating, the means for employing and the means for executing comprise automated means.

23. (Original) The system of claim 20, wherein the means for executing comprises means for executing in parallel at least some of the test case execution threads for multiple layers of the software component.

24. (Original) The system of claim 23, wherein the means for executing comprises means for executing in parallel at least some test case execution threads of each layer of the software component.

25. (Original) The system of claim 20, wherein the means for separating the test cases based on layers of the software component results in each layer of the software component having multiple test cases associated therewith.

26. (Original) The system of claim 20, wherein the abstraction matrix describes each layer of the software component.

27. (Original) The system of claim 26, wherein the abstraction matrix comprises a separate abstraction file for each layer of the layers of the software component.

28. (Original) The system of claim 20, further comprising means for providing attributes for use in generating the test cases and mapped expected results therefore.

29. (Original) The system of claim 20, wherein the abstraction matrix comprises state information and events that are associated with each state of the software component.

30. (Original) The system of claim 29, wherein the abstraction matrix includes current state information and next state information, and wherein for ate least one current state the event information comprises an event which leads from that current state to at least one next state.

31. (Original) The system of claim 20, wherein the software component comprises an operating system component.

32. (Original) The system of claim 20, wherein the abstraction matrix identifies an minimal number of states required to model the software component.

33. (Original) The system of claim 20, wherein the means for associating data structures for the separated test cases of the layers comprises means for associating information on the relationships between layers, thus allowing test cases of the various layers to be uncorrelated.

34. (Currently Amended) A computer-implemented system for testing a software component comprising multiple layers of software, the system comprising:

an abstraction matrix in mathematical abstract form that describes the software component, the abstraction matrix partitioning the software component into multiple software layers and comprising state and event information taking into account relationships that exist between the multiple software layers;

[[an]] a computer-implemented abstraction engine for parsing the abstraction matrix and automatically generating and factoring out doable test cases and mapped expected results therefor;

wherein the abstraction engine separates the test cases based on the software layers of the software component, and associates data structures with the separate test cases of the software layers, the data structures allowing the test cases of the various software layers to be uncorrelated; and

a computer-implemented functional verification test engine adapted to take the software component in executable form and generate for each software layer of the software component test case execution threads from the test cases and mapped expected results for that software layer, wherein the functional test verification engine outputs in parallel at least some test case execution threads for at least one software layer of the software component, thereby allowing testing of the software component.

35. (Currently Amended) A system for generating test cases for use in testing a software component, the software component comprising multiple layers of software, said system comprising:

a functional specification of the software component;

an abstraction matrix in mathematical abstract form that describes the software component, the abstraction matrix partitioning the software component into multiple software layers and comprising state and event information taking into account relationships that exist between the multiple software layers, and having been created from the functional specification;

means for parsing the abstraction matrix to automatically generate and factor out doable test cases and mapped expected results therefor; and

means for separating the test cases based on the software layers of the software component, and for associating data structures with the separated test cases of the software layers, the data structures allowing the test cases of the various ~~layer~~ software layers to be uncorrelated.

36. (Original) The system of claim 35, wherein the means for parsing and the means for separating comprise automated means.

37. (Original) The system of claim 35, wherein the abstraction matrix comprises a separate abstraction file for each layer of the layers of the software component.

38. (Original) The system of claim 35, wherein the abstraction matrix includes current state information and next state information, and wherein for at least one current state the event information comprises an event which leads from that current state to at least one next state.

39. (Original) The system of claim 35, wherein the means for associating data structures for the separated test cases of the layers comprises means for associating information on the relationships between layers, thus allowing test cases of the various layers to be uncorrelated.

40. (Currently Amended) A computer-implemented system for generating test cases for use in testing a software component comprising multiple layers of software, the system comprising:

a storage medium for storing an abstraction matrix in mathematical abstract form that describes the software component by partitioning the software component into multiple software layers, the abstraction matrix, which comprises state and event information taking into account relationships that exist between the multiple software layers, having been created from an automatic partitioning of a functional specification of the software component;

[[an]] a computer-implemented abstraction engine for automatically retrieving and parsing the abstraction matrix to automatically generate and factor out doable test cases and mapped expected results therefor; and

wherein the abstraction engine separates the test cases based on the software layers of the software component and associates data structures with the separated test cases of the software layers, the data structures allowing the test cases of the various software layers to be uncorrelated.

41. (Currently Amended) At least one program storage device readable by machine, tangibly embodying at least one program of instructions executable by the machine to form a method for testing a software component comprising multiple layers of software, the method comprising:

storing an abstraction matrix in mathematical abstract form that describes the software component by partitioning the software component into multiple software layers, the abstraction matrix comprising state and event information taking into account relationships that exist between the multiple software layers;

parsing the abstraction matrix to automatically generate and factor out doable test cases and mapped expected results therefor;

separating the test cases based on the software layers of the software component, and associating data structures with the separated test cases of the software layers, the data structures allowing the test cases of the various software layers to be uncorrelated;

employing the software component in executable form to generate for each software layer of the software component test case execution threads from the test cases and mapped expected results for that software layer; and

executing in parallel at least some of the test case execution threads for at least one software layer of the software component, thereby testing the software component.

42. (Original) The at least one program storage device of claim 41, wherein the executing comprises executing in parallel at least some of the test case execution threads for multiple layers of the software component.

43. (Original) The at least one program storage device of claim 42, wherein the executing comprises executing in parallel at least some test case execution threads of each layer of the software component.

44. (Original) The at least one program storage device of claim 41, wherein the separating the test cases based on layers of the software component results in each layer of the software component having multiple test cases associated therewith.

45. (Original) The at least one program storage device of claim 41, wherein abstraction matrix describes each layer of the software component.

46. (Original) The at least one program storage device of claim 45, wherein the abstraction matrix comprises a separate abstraction file for each layer of the layers of the software component.

47. (Original) The at least one program storage device of claim 41, wherein the abstraction matrix comprises state information and events that are associated with each state of the software component.

48. (Original) The at least one program storage device of claim 47, wherein the abstraction matrix includes current state information and next state information, and wherein for at least one current state the event information comprises an event which leads from that current state to at least one next state.

49. (Original) The at least one program storage device of claim 41, wherein the software component comprises an operating system component.

50. (Original) The at least one program storage device of claim 41, wherein the associating data structures for the separated test cases of the layers comprises associating information on the relationships between layers, thus allowing test cases of the various layers to be uncorrelated.

51. (Currently Amended) At least one program storage device readable by a machine, tangibly embodying at least one program of instructions executable by the machine to perform a method for generating test cases for use in testing a software component comprising multiple layers of software, the method comprising:

storing an abstraction matrix of the software component in mathematical abstract form that describes the software component by partitioning the software component into multiple software layers, the abstraction matrix comprising state and event information taking into account relationships that exist between the multiple software layers;

parsing the abstraction matrix to automatically generate and factor out doable test cases and mapped expected results therefor; and

separating the test cases based on the software layers of the software component, and associating data structures with the separated test cases of the software layers, the data structures allowing the test cases of the various software layers to be uncorrelated.

52. (Original) The at least one program storage device of claim 51, wherein the abstraction matrix comprises a separate abstraction file for each layer of the layers of the software component.

53. (Original) The at least one program storage device of claim 51, wherein the abstraction matrix includes current state information and next state information, and wherein for at least one current state the event information comprises an event which leads from that current state to the at least one next state.

54. (Original) The at least one program storage device of claim 51, wherein the associating data structures for the separated test cases of the layers comprises associating information on the relationships between layers, thus allowing test cases of the various layers to be uncorrelated.

55. (New) The method of claim 1, wherein the data structures for at least some software layers mimic information passed between layers during normal operation of the software component.

56. (New) The method of claim 55, wherein the abstraction matrix comprises an abstraction file for each software layer, and wherein the parsing comprises generating a mapped expected result for each line of each abstraction file based on associated state information.

57. (New) The method of claim 56, wherein the separating comprises eliminating dependencies between at least one input of a software layer based on other inputs of other software layers of the software component, thereby uncorrelating test cases of the various software layers.

58. (New) The method of claim 57, wherein the software component comprises a cluster operating system component.